Proof Hints for Event-B

Thai Son Hoang

Institute of Information Security, ETH Zurich

Abstract. Interactive proofs are often considered as costs of formal modelling activity. In an incremental development environment such as the Rodin platform for Event-B, information from proof attempts is important input for adapting the model. This paper considers the idea of using interactive proofs to "improve" the model, in particular, to convert them into automatic ones. We propose to lift some essential proof information from the interactive proofs into the model as what we called *proof hints*. In particular, proof hints are not only for the purpose of proofs: it helps to understand the formal models better.
Keywords: Event-B, formal modelling, proof hints, the Rodin platform

1 Introduction

Event-B is a refinement-based modelling method, which can be used to develop various types of systems. Starting with an abstract specification, several refinement steps gradually introduce more details into the formal models in a consistent manner. An important part of an Event-B formal model is the verification conditions generated as proof obligations. The task of discharging these obligations is first given to some automated provers. Afterwards, remaining undischarged proof obligations are required to be proved interactively. Typically, manual proofs are considered as "costs" of development, given the required human interaction for produced them, and the difficulty in maintaining them when the formal models evolve.

As the size of models grows, the complexity of the associated proof obligations also increases, hence interactive proofs are inevitable. Improving the performance of the automatic provers has been considered with some success [3,4,2]. Despite the improvement in the percentage of automatic proofs, interactive proofs are still an obstacle in developing and maintaining formal models.

In this paper, we attempt to answer the following question. *Can we improve our formal models in such a way that helps the proofs?* After all, modelling using refinement is also a way of structuring the proof of correctness of the models. We propose some notions to encapsulate essential proof details extracted from interactive proofs within the formal model. We call the additional information to the formal models "proof hints".

Some form of *proof hints* already exists in Event-B, *e.g.*, "witnesses" or "theorems". These useful features are designed not only to help with proving the correctness of the model but also to give more information about the particular

model, i.e., *why it is correct*. In fact, "proof hints" should help to understand the formal model better.

We consider the current state of *Rodin Platform* (*Rodin*) and show two kinds of useful proof information that can be included in the formal models, namely, to *select hypotheses* and to *perform a proof by cases*.

Select hypotheses Indicates that some facts (e.g., invariants or axioms) are required for discharging the obligations.

Perform a proof by cases Indicates that the proof can be discharged by consider different cases.

We show that the effect of the proof hints can be "simulated" at the moment by some modelling "tricks" in Event-B.

In the long term, we propose to have an extension to Event-B and to *Rodin*, to have proof hints as a part of the model and to implement a plug-in for interpreting the proof hints and applying these hints appropriately during proofs.

Organisation. The rest of the paper is structured as follows. Section 2 gives some background on Event-B and *Rodin*. Section 3 presents our ideas of proof hints. Section 4 illustrates proof hints by means of two examples. Section 5 discusses some proposals for the tool support. We give some conclusions in Section 6.

2 Background

2.1 The Event-B Modelling Method

An Event-B model corresponds to a discrete transition system and is divided into two parts: a *static* part called *context* and a *dynamic* part called *machine*. A context may contain *carrier sets* (types), *constants*, *axioms* (assumptions about sets and constants). For clarity, we omit references to context in the sequel.

Machines may contain *variables*, *invariants*, and *events*. Variables v define the state of a machine and are constrained by invariants $I(v)$. An event e can be represented as $\mathsf{e} \;\widehat{=}\; \textbf{any}\ x\ \textbf{where}\ G(x,v)\ \textbf{then}\ Q(x,v)\ \textbf{end}$, where x stands for the event's *parameters*, thus allowing for state changes. The *guard* $G(x,v)$ states the necessary condition under which an event may occur. The *action* $Q(x,v)$ describes how state variables v evolve when the event occurs. The short form $\mathsf{e} \;\widehat{=}\; \textbf{when}\ G(v)\ \textbf{then}\ Q(v)\ \textbf{end}$ is used when the event does not have any parameters, and we write $\mathsf{e} \;\widehat{=}\; \textbf{begin}\ Q(v)\ \textbf{end}$ when, in addition, the event's guard equals *true*. A dedicated event in the last form is used for the *initialisation* event (init). The action of an event is composed of one or more *assignments* of the form: $v := E(x,v)$, $v :\in E(x,v)$, $v :| P(x,v,v')$, specifying v becomes $E(x,v)$, v becomes an element of $E(x,v)$, and v becomes such that $P(x,v,v')$ holds, respectively. All assignments of an action $Q(x,v)$ occur *simultaneously*. As a result, each event e is associated with a before-after predicate, denoted as $\boldsymbol{Q}(x,v,v')$.

The *invariant preservation* proof obligation (INV) states that invariants are maintained whenever variables are updated. For each event e, the corresponding proof obligation is as follows.

$$I(v), G(x, v), \boldsymbol{Q}(x, v, v') \vdash I(v') \quad \text{(INV)}$$

All predicate modelling elements, *e.g.*, axioms, invariants, guards, can be also declared as theorems. Theorems need to be proved when they are declared. As an example, a theorem in guard must be proved to be a consequence of axioms, invariants, and previously declared guards.

Machine refinement is a mechanism for introducing details about the dynamic properties of a model [1]. The states of the abstract machine M are related to the states of the concrete machine N by *gluing invariants* $J(v, w)$, where v and w are the variables of the abstract and concrete machine, respectively. Each event e of the abstract machine is *refined* by a concrete event f (later we will relax this constraint). Assume that the concrete event is of the following form f $\widehat{=}$ **any** y **where** $H(y, w)$ **then** $R(y, w)$ **end** . Somewhat simplifying, we can say that e refines f if the gluing invariant establish a simulation of f by the e. This is presented as the following obligation.

$$I(v), J(v, w), H(y, w), \boldsymbol{R}(y, w, w') \vdash \exists x, v' \cdot G(x, v) \wedge \boldsymbol{Q}(x, v, v') \wedge J(v', w') \quad \text{(REF)}$$

In order to split the above proof obligation, Event-B introduces the notion of "witnesses" for x and v'. The witnesses are predicates $W_1(x, y, v, w, w')$ (for x) and $W_2(v', y, v, w, w')$ (for w'), which are required to be *feasible*, i.e., satisfying the following proof obligations.

$$I(v), J(v, w), H(y, w) \vdash \exists x \cdot W_1(x, y, v, w, w') \quad \text{(WFIS1)}$$

$$I(v), J(v, w), H(y, w), \boldsymbol{R}(y, w, w') \vdash \exists v' \cdot W_2(v', y, v, w, w') \quad \text{(WFIS2)}$$

The witnesses supply instances of x and v' (provided that they exist) for instantiating the proof obligation REF. Given the witnesses, the refinement proof obligation REF can be split into the following proof obligations.

$$I(v), J(v, w), H(y, w), W_1(x, y, v, w, w') \vdash G(x, v) \quad \text{(GRD)}$$

$$I(v), J(v, w), H(y, w), \boldsymbol{R}(y, w, w'), W_1(x, y, v, w, w'), W_2(v', y, v, w, w') \vdash \boldsymbol{Q}(x, v, v') \quad \text{(SIM)}$$

$$I(v), J(v, w), H(y, w), \boldsymbol{R}(y, w, w'), W_1(x, y, v, w, w'), W_2(v', y, v, w, w') \vdash J(v', w') \quad \text{(INV)}$$

In the course of refinement, *new events* are often introduced into a model. New events must not modify abstract variable v. When an abstract event e is refined by more than one concrete events f, we say that the abstract event e is *split* and prove that each concrete f is a valid refinement of the abstract event. A concrete event f can refine two (or more) abstract events e, provided that the abstract events are only different in guards. We say that the abstract events are *merged* into the concrete event f. It is required to prove that the guard of f is stronger than the disjunction of the guards of the abstract events.

2.2 Proving with the Rodin Platform

Rodin is an Eclipse-based tool chain for analysing and reasoning about Event-B models. Models are developed incrementally within the platform. Two main activities of developers are *modelling* and *proving* (as illustrated in Fig. 1). Proof obligations are generated from modelling and are input to the proving activity. Information about proof attempts from proving are input to the modelling activity to "improve" the models. In particular, failed proof attempts usually indicate some problems with the models and give hints on how the models can be fixed, *e.g.*, to strengthen the guard of some events or to add some missing invariants into the models.

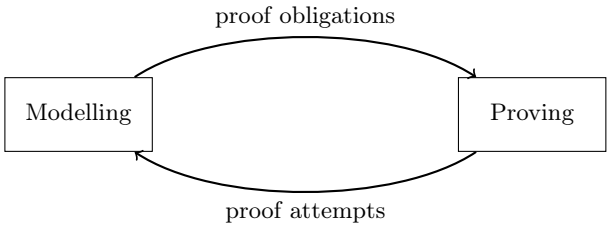


Fig. 1. Developing Event-B models using *Rodin*

Obligations are proved either automatically or manually. In automatic mode, *Rodin* uses some predefined proof tactics made up of internal and external provers to discharge the obligations. In interactive mode, the user "guides" the proof attempts by applying some simple proof steps to simplify the obligations before invoking some trusted external provers to finish the proofs. As interactive proofs require manually intervenions, it is usually considered as some costs of developing formal models. Moreover, maintenance of interactive proofs is difficult: a change in the formal model more often invalidates the interactive proofs. As a result, improving the rate of automatic proofs will also help to maintain the models better.

We consider some common interactive proof steps, e.g., to *add hypothesis*, to *select hypotheses*, and to *perform a proof by cases*.

Add hypothesis This proof step corresponds to the following proof rule.

$$\frac{\mathbf{H} \vdash P \quad \mathbf{H}, P \vdash G}{\mathbf{H} \vdash G}\ \textbf{CUT}$$

The rule allows add P as a hypothesis, provided that it can be proved from the current hypotheses $\mathbf{H}$.

Select hypotheses *Rodin* has a notion of *selected* hypotheses which is used by some external provers. Often, too many irrelevant hypotheses will have

negative effect on the performance of external provers. By restricting the set of hypotheses available to these external provers, the user helps the external provers to concentrate on using only some relevant hypotheses. An example for selected hypotheses are guards of an event: they are by default selected for proof obligations related to the event.

Perform a proof by cases This proof step allows user to perform a proof by cases, with respect to some condition P.

$$\frac{\mathbf{H}, P \vdash G \quad \mathbf{H}, \neg P \vdash G}{\mathbf{H} \vdash G}\ \textbf{CASE}$$

The proof is split into two branches accordingly.

3 Proof Hints

There are existing work for improving the rate of automatic proofs. Recently, some links to external provers such as Isabelle [4], SMT [2] have been added to *Rodin*. Selected hypotheses can be calculated according to some heuristic [3]. However, interactive proofs are still unavoidable. We look at the problem from a different angle: to convert interactive proofs into automatic proofs by improving the formal models, essentially exposing some proof information in the formal models. We call these additional proof information *proof hints*.

There is already several such proof information existing in the Event-B models, normally being seen as part of the model rather than some exposed proof information.

- Theorems in the model is a special case of *adding a hypothesis* in an interactive proof.
- Automatic *selection of guards* for the event's proof obligations.
- The witnesses can be seen as some hints for manually *instantiating* the existential goal of the general proof obligation REF, which results in three sub-obligations GRD, SIM, and INV.

In principle, any proof information can be lifted to be proof hints, part of the model. However, revealing the actual proof is certainly undesirable: this could have negative effects on the understanding of the model. In fact we want only to exposing *essential* information about the proofs. We believe that the criteria for proof hints should be as follows.

1. They should help to *automate* more proofs.
2. They should help to better *understand* the model.

While the first criteria is straightforward, our emphasis is on the second criteria. Once again, low level proof information is irrelevant for understanding of the model. We only want to have essential key important proof steps as hints, in order to justify about the correctness of the formal models.

4 Some Useful Proof Hints

This section presents two kinds of proof hints, namely to *select hypotheses* and to *perform a proof by cases*.

4.1 Select Hypotheses

During an interactive proof section, the developer can complete the proof by selecting some hypotheses and invoke one of the provers that uses only selected hypothesis, *e.g.*, AterlierB P0. The solution to make the proof become automatic is to (somehow) give hints to *Rodin* to select these additional hypotheses automatically.

Example Consider the following specification containing two variables x and y. The machine has a single event[1] called set, which assign $y+1$ to x when x is either 1 or 2.

variables: x, y

invariants:
hypSel0_1 : $x \in \mathbb{N}$
hypSel0_2 : $x \neq 0 \Rightarrow y \in \mathbb{N}$

set
when
grd1 : $x \in \{1, 2\}$
then
act1 : $x := y + 1$
end

We are interested in proof obligation set/**hypSel0_1**/INV stating that event set maintains the invariant **hypSel0_1**.

$$x \in \mathbb{N}, x \neq 0 \Rightarrow y \in \mathbb{N}, x \in \{1, 2\} \vdash y + 1 \in \mathbb{N}$$

The proof obligation cannot be discharged automatically. In particular, by default, the selected hypotheses are **hypSel0_1** and **grd1**. The obligation can be discharged by selecting **hypSel0_2**, and invoke external provers using selected hypotheses, such as AterlierB P0.

Workaround A simple workaround to have **hypSel0_2** being selected automatically is to add the invariant as a *theorem* in guard of set.

set
when
grd1 : $x \in \{1, 2\}$
thm1 : $x \neq 0 \Rightarrow y \in \mathbb{N}$
then
act1 : $x := y + 1$
end

[1] For clarity we omit the initialisation event init.

The additional theorem can be removed in further subsequent refinement if necessary.

Proposal The disadvantages of the above approach are as follows.

- This introduces extra proof obligations to prove that the copies of the invariants are theorems in guard (even though those proof obligations are discharged automatically).
- Recopying the text of the invariants is error-prone.
- Reformulating invariants leads to the need for changing the text of the extra theorems.

Our proposal is to have a specific "proof hint" for events. This form of proof hints will specify the invariant/theorem need to be used automatically. For example, we could extend event set with the following hint of using **hypSel0_2** for the maintenance of **hypSel0_1**.

set
  when
    $x \in \{1, 2\}$
  then
    $x := y + 1$
  hints
    use **hypSel0_2** for **hypSel0_1**
  end

4.2 Perform a Proof by Cases

Sometimes, during an interactive proving session, the user suggests a predicate P in order to do a "proof by cases". The subsequent branches of the proof can be discharged easily. It is hence desirable to have this hint about performing a proof by cases in the model. Automatic provers often do not apply the case splits automatically, since this potentially leads to blow up in terms of the number of sub-goals.

Example Consider the following specification with three variables a, b, c.

variables: a, b, c

invariants:
  case0_1 : $a \leq c$
  case0_2 : $a \neq 1 \Rightarrow b = a + 1$
  case0_3 : $a = 1 \Rightarrow b \leq c$

set
  begin
    $a := b - 1$
  end

The interesting proof obligation to look at is set/**case0_1**/INV.

$$a \leq c, a \neq 1 \Rightarrow b = a + 1, a = 1 \Rightarrow b \leq c \ \vdash \ b - 1 \leq c$$

Informally, the reasoning follows the cases of either $a = 1$ or $a \neq 1$ and applying **case0_2**, **case0_3** accordingly. Hence we would like to give the hints about the case split. The obligation is not discharged by the default automatic prover within *Rodin*.

Workaround In order to "simulate" the effect of introducing this proof hints, we first split the event into two sub-events, guarded by corresponding conditions.

$$\text{set\_case1} \ \widehat{=} \ \textbf{when} \ a = 1 \ \textbf{then} \ a := b - 1 \ \textbf{end}$$
$$\text{set\_case2} \ \widehat{=} \ \textbf{when} \ a \neq 1 \ \textbf{then} \ a := b - 1 \ \textbf{end}$$

The original event set can be obtained by merging the above two events using refinement.

```
set
  refines  set_case1, set_case2
  begin
    a := b − 1
  end
```

which leads to a trivial proof obligation to prove about merging events.

Proposal There are several disadvantages of the workaround:

- Splitting of event and merging using refinement is artificial.
- Splitting of events leads to double number of proof obligations (those that does not need the case split).

Our proposal is to provide a hint directly in the model about the case split.

```
set
  begin
    a := b − 1
  hints
    split case using a = 1 for case0_1
  end
```

5 Ideas on Tool Support

Given the extensibility of *Rodin*, having proof hints as additional elements of Event-B models would be straightforward. How the hints are interpreted and work with the automatic provers of *Rodin* is a more challenging topic. There are some options for the implementation of the "hint-interpreter".

The first option is to have the interpreter to effect the generated proof obligations. For example, two different proof obligations are generated according to the "proof-by-cases" hint to replace the original proof obligation. This requires to alter the *Proof Obligation Generator* (*POG*) of *Rodin* to take into account the hints. The second option is to leave the original proof obligations untouched and to incorporate the hints at the start of a proof, *i.e.*, they can be applied before the automatic provers are invoked.

At the moment, we are investigating these options for tool support. The first option is similar to witnesses in event refinement, to split proof obligation REF into obligations GRD, SIM, and INV. As a result, it changes the generated proof obligations of Event-B models, which might be undesirable. In particular, the number of proof obligations generated can be different depending on whether proof hints are present. The second option applying proof hints at the beginning of each proof, works for the two illustrated proof hints presented within this paper. In general, we might want to have more general proof hints that should be apply in the middle of a proof, or even combining different hints in one proof.

6 Conclusion

We presented some ideas about the notion of "proof hints" for Event-B and discusses the possibilities of extending the supporting *Rodin platform*. In a broad term, proof hints essentially are proof information that are added to the model. We proposed two kinds of proof hints in this paper, for suggesting selected hypotheses and performing proof-by-cases. We presented some workaround at the moment to "simulate" the proof hints and to automate some proofs in the current version of *Rodin*. The illustrated examples are simple and seem to be unrealistic. However, they are extracted from some large development (adapted accordingly). Their simplicity is not too illustrate the weakness of *Rodin*'s automatic provers, but rather to support the argument that formal proofs of systems are challenging tasks.

Often when describing an Event-B formal model, we also need to explain why the model is correct, *e.g.*, why guard strengthening or invariant preservation is satisfied. Proof hints should give some information (but not too much) to answer the questions about the correctness of models. It would be the ultimate goal of having self-explained formal models, not only in terms of how they works (*e.g.*, events) and what their properties are (*e.g.*, invariants), but also why they are correct.

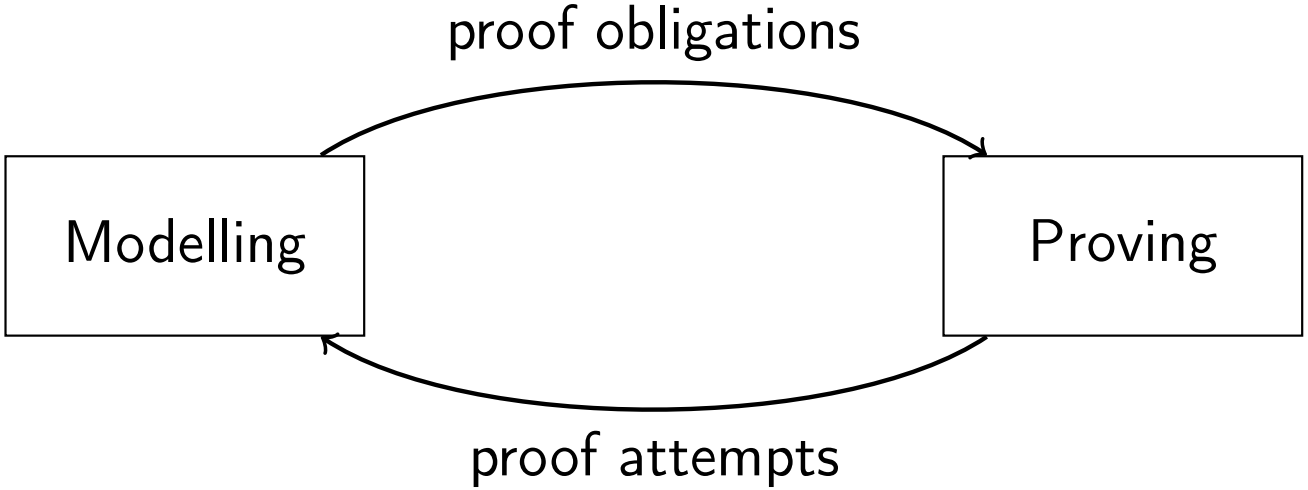
We do not propose proof hints as a way to avoid interactive proofs. More often, we need to perform some interactive proof steps, in order to figure out or understand why the obligation can be discharged. The role of proof hints therefore is to convert some interactive proofs into automatic ones, helping the model to be more resilient to changes.

In the long term, we might want to extract the essence of proofs as some high-level structured proofs (similar to Isabelle/Isar [5]). This requires more

investigation in terms of the usefulness of such an approach, and subsequent tool support.

References

1. J-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, May 2010.
2. David Déharbe, Pascal Fontaine, Yoann Guyot, and Laurent Voisin. Smt solvers for rodin. In John Derrick, John A. Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene, editors, *ABZ*, volume 7316 of *Lecture Notes in Computer Science*, pages 194–207. Springer, 2012.
3. Jann Röder. Relevance filters for Event-B. Master's thesis, ETH Zurich, 2010. http://e-collection.library.ethz.ch/view/eth:2278.
4. Matthias Schmalz. Export to Isabelle plug-in. http://wiki.event-b.org/index.php/Export_to_Isabelle, 2011.
5. Markus Wenzel. *The Isabelle/Isar Reference Manual*. Technische Universität München, 2002. http://isabelle.in.tum.de/dist/Isabelle2002/doc/isar-ref.pdf.


proof obligations
Modelling
Proving
proof attempts